Abstract:

The rising cost of support and maintenance for desktop systems has fueled a growing interest in thin-client computing. Although most modern operating systems provide powerful abstractions to applications, the single interface they provided can’t meet the requirements of daedal applications. Especially for thin-client systems, the interface has seriously limited the performance of special applications and exposes more system services to applications than they need, which makes against the system security. In this paper, multiple-interface operating systems are presented. They could provide the special applications with specific interface while other applications still use general interface as usual. Designers can grant the application a privilege to a set of interfaces. The multiple-interface operating systems for thin-client platforms are characteristic of multiple-interface and monolithic-kernel. The multiple-interface offers feasibility and adaptability while the monolithic-kernel guarantees the performance. A prototype of this system has been implemented based on Linux 2.4.17 for network computers in our lab. We believe that these operating systems are a better solution to the thin-client platforms than traditional unitary-interface operating systems, enabling them provide efficient and secure services to diverse applications.

Keywords: unitary-interface, multiple-interface, monolithic-kernel, operating systems, thin-client

1 Introduction

During the last two decades, the centralized computing model of mainframe computing has shifted to a distributed model of desktop computing. But expensive supports for the desktop system set up a huge barrier in front of us. This leads to a more centralized and easier-to-manage computing model, thin-client model, where clients are not only responsible for displaying the results of server operations as traditional ones, but also supporting some applications locally.

Indubitably, operating systems play a crucial role in the effectiveness and efficiency of the applications on the thin-client platforms. Many modern thin-client systems are designed to provide the complex graphical interfaces and support applications previously available on desktop systems while centralizing computing works on powerful servers to reduce administration costs and make more efficient use of shared computing resources. Examples of popular thin-client systems include Microsoft Terminal Services, AT&T VNC, Citrix MetaFrame, SUN Ray and so on. The performance of these operating systems has been studied [1]. The result suggests that the performance of the same application is greatly dependent on the underlying operating systems.

However, the interface in existing operating systems, for either personal computers or thin-client platforms mentioned above, is fixed and appears the same to all applications [2][3]. This points that the key problem operating systems facing is how to support efficiently a range of applications. But the single interface (named unitary-interface in this paper) is problematic in the face of diverse applications of thin-clients platforms. First, all the applications running on the system use the same set of services and implementations defined by the interface which must compromise between efficiency and compatibility [4][5]. It has seriously limited the performance, flexibility and functionality of applications on thin client systems since it fails to provide the best utilization of the physical resources to both general applications and special applications (browser, multimedia etc.). Second, the single interface grants privilege with coarse granularity, so applications acquire more privileges than they require which goes against the principle of least privilege [6]. To address these problems, we propose operating systems with multiple-interface mechanism on the basis of monolithic-kernel and a prototype is implemented based on Linux 2.4.17.

The remaining sections are organized as follows. Section 2 introduces the traditional unitary-interface and investigates its problems. Section 3 presents prior solutions to unitary-interface and related works of multiple-interface. In Section 4, basic conception of multiple-interface and monolithic-kernel system we proposed is described and some detailed discussion is made. Conclusion and future works are in Section 5.

2 The unitary-interface

As mentioned above, an operating system provides an abstract view of the underlying hardware to all applications. It is the lower-level software which user’s programs run on. Together they form a two-level structure that the upper level is implemented by using the interface implemented by the lower level. The primary benefits of two-level implementation are modularity, high-level programmability and reusability [7]. It is widely used in the operating system design and other domains. The design of the interface of the two levels is essential to the communication efficacy.

Traditional operating system designers define a fixed set of flat interfaces which provide the same set of services to all applications. This interface absolutely separates the upper level and the lower level. It is called the unitary-interface mechanism. The unitary-interface mechanism has dominated the operating system domain for so long time and worked well. This approach, however, is now problematic and showing its limitations in the face of diverse applications of thin-client
There are two intrinsical and essential features of the unitary-interface mechanism:

- **Separateness of adjacent levels**: In order to get the services of the lower level, programs of the upper level must use data types and functions defined by the interface. Therefore, this interface separates these two adjacent levels. As to operating systems, applications can’t execute kernel codes, or modify kernel data directly, but only can do the things through the system calls interface.
- **Interfacial singleness**: Two-level implementation has the benefit of reusability, and the lower level can be reused by more than one upper levels. However, in operating systems domain, all upper levels have the same view of the lower level, since only one interface is defined. Concretely, all applications share the same set of system calls.

The traditional “all-in-one” interface has seriously limited the effectiveness and efficiency of specific applications on the modern thin-client systems because the interface provided to all applications is truly single-adaptable, and it’s hard to be adjusted or optimized for the specific applications\[2\]. In addition, applications have privilege to more system services than they need, which may be dangerous to the whole system [6]. So solutions must be found to get over these problems.

### 3 Solutions to unitary-interface mechanism

The researchers of operating systems have long realized the problem of the unitary-interface mechanism and many solutions have been brought forward, such as:

- **Vertical-structure operating systems**: Viewed from the structure, vertical-structure operating systems are a set of stackable extended machines. In such an operating system, applications could choose or build their own in-between levels to provide the necessary or specific services. The MIT’s Exokernel architecture [8], and Stanford’s Cache kernel [9] are probably the best-known examples of such operating systems.

- **Micro-kernel operating systems**: In a micro-kernel operating system, the kernel itself implements only the minimum of virtualization services, delegating the provision of services such as networking and file systems to server processes. By virtue of supporting multiple servers, each providing a different set of services, the operating system can offer different resource management strategies to different applications, thus avoiding the “one-size-fits-all” problem. Mach [10] perhaps typifies the micro-kernel architecture.

- **Extensible kernels**: While vertical-structure operating systems could provide application-specific resource management by exporting low-level interfaces which applications can use to implement their own policies, an alternative approach is to allow applications to extend the kernel with functions which implement those policies. Extensible systems like VINO [11] and SPIN [3] allow applications to replace OS functionality with new implementations, making what was once a run-time invariant now variable under applications’ control.

All these solutions could support applications with more flexibility than traditional monolithic-kernel operating systems, and enhance the performance of specific applications to a certain extent. However, the essence of the unitary-interface mechanism does hold and other problems may arise. In vertical-structure operating systems, each extended machine still only provides one stable interface to other levels or applications. Most of the problems induced by unitary-interface have not been eliminated, such as interfacial completeness, span-interface optimization, and interfacial complexity. In micro-kernel operating systems, both applications and server processes run on the kernel, and the better part of unitary-interface problems is avoided. But much more work should be done to implement and maintain various and perhaps independent server processes. In extensible kernels, the “all-in-one” interface problem is not solved though some internal policies can be replaced by applications’. Moreover, the range of replaceable policies in kernel is greatly limited, which must be decided during the kernel design. And the designers must pay attention to the stability and security of the systems.

While all kernels of these systems preserve the traditional “all-in-one” interface, we believe that the multiple-interface mechanism is a good way to avoid the problems raised by unitary-interface one. Indeed, the multiple-interface mechanism had been introduced in object-oriented methodology in 1980s[12]. Also, it is applied in some component-based operating systems [13] [14]. However, these operating systems don’t utilize the multiple-interface between the whole kernel and the upper applications, but in-between the objects or components inside the kernel. In other words, it doesn’t change the fact that the applications still only have the same one interface of the kernel. Considering, we suppose that the multiple-interface mechanism should be used between the kernel and the applications to meet the requirement of thin-client platforms.

### 4 Multiple-interface and monolithic-kernel

In this section, the multiple-interface and monolithic-kernel operating system designed for thin-client is described in detail and several related mechanisms in the system are stated.

#### 4.1 The operating system model

Multiple-interface and monolithic-kernel are two characteristics of the operating system designed for thin-client platforms in this paper. As a simple example, we suppose that a thin-client is used in the environment that browser and multimedia are the primary applications, and other general applications are also available, such as ssh and emacs. How should the operating system be?

Usually, an operating system provides a set of services to applications, mostly data types (e.g. files) and operations on them (e.g. read). Together, they form the interface for the applications running on the system. In traditional unitary
interface operating systems, only one interface is defined, and the interface is the same to all applications, as shown in Figure 1(a). However, in view of the two special applications, special interfaces for them should be considered. For the browser, system calls of network and file system could be specialized and the implementation could be optimized specifically. Thus, the performance may be enhanced. For the multimedia, the applications could use the power of the video and audio decoder if available, and of course, these services are excluded by a general-purpose interface. A multiple-interface operating system for this thin-client can be designed as shown in Figure 1(b). Specific interface I is defined for the browser applications and specific interface II is defined for the multimedia applications. Besides, a general interface is provided to support common applications. Compared to the unitary-interface in Figure 1(a), the multiple-interface mechanism exhibits better adaptability to the thin-client systems.

Another key issue in the operating system design is the structure of the kernel. As a rule, modern operating systems must find its own solution to the conflict between performance and modularity, and in embedded systems, the performance is of vital importance, so the monolithic-kernel is the most pervasive choice. In a monolithic kernel, each procedure inside is free to call any other one (as opposed to the structures containing modules, packages or layers, in which much of the information is hidden away, and only the officially designated entry points can be called from outside), which makes the kernel to be implemented without any excrecent performance cost for reason of the complex structures. Consequently, the multiple-interface operating systems for the thin-client systems also use monolithic kernel structure to guarantee the performance.

![Figure 1: Unitary-interface and Multiple-interface](image)

4.2 Interface design

The best place to begin an operating system design is to think about the interface it provides. While in the multiple-interface operating systems, the designers have to determine how many interfaces should be included and what should be defined for each interface. At the same time, each interface definition may as well accord with the philosophy of interface design: simplicity, completeness, and efficiency.[15]

The applications on the thin-client could be classified into different types according to the importance or frequency of usage. Then, different interfaces can be provided to different types of applications to satisfy their necessary constraints. As to the sample mentioned above, there are three types of applications. The internet browser is very essential, and it demands particular optimizations in order to get higher performance. The multimedia applications need to use hardware decoder to release the processor. In addition, other common applications, such as ssh and emacs, are required to run at times. So, three interfaces should be defined, one specific interface for the browser, another specific interface for the multimedia applications, and one general interface for the others.

The number of interfaces must be strictly controlled in that it perhaps increases the complexity and affects the stability of the kernel badly. Each interface and its relevant implementation must aim at its own goal, for example, performance or exposure of certain hardware. One interface is added only if it is necessary. In our opinion, two or three specific interfaces are appropriate.

Though there are many interfaces in this operating system, each interface should accord with the three design principles. To the general interface, it should be based on a standard, such as POSIX[16] or uITRON [17]. While to the specific interfaces, their definitions and relevant implementations could more stand to the three principles since each interface’s goal is unambiguous and unambitious.

4.3 Access control

The multiple-interface can provide applications with flexible system services. Also, designers can grant application a privilege to a set of interfaces. Attaching access control to the interface offers the thin-client a simpler protection strategy and increases the reliability of the system.

In multiple-interface operating systems, this protection strategy could be more flexible and easier to be implemented by defining the map between interfaces and applications. Many approaches to this problem have been raised in object-oriented domain[12], and two methods are described as follows, which are believed more proper to the operating systems.

- **Membership Sets:** The most intuitive implementation is to maintain an invocation table of the system. An entry of the table consists of a process identifier and an interface identifier. To check that a service required by a process is valid, the underlying system must look up the invocation table to see whether it contains the suited entry.
- **Keys:** Each process and interface has its key, and the system determines a matching function. When a process calls a service, the underlying system must consult the matching function with the two keys as input. Key-based validity checking is much more efficient than other means.

4.4 Summary

In virtue of the two characteristics, multiple-interface and monolithic-kernel, the problems caused by unitary-interface on
thin-client systems are eliminated. Monolithic-kernel guarantees the performance, and multiple-interface offers feasibility and adaptability. In summary, there are three major advantages of this operating system compared to a traditional one: adaptability, efficiency, and security.

On the other hand, some problems must be paid attention to when the multiple-interface and monolithic-kernel operating systems is used. The greatest one is the complexity of the kernel, since it directly affects the reliability and stability of the system. Also, portability and security are important. These matters are not obstructive to the use of these operating systems in the thin-client platforms. Distinguished from personal computer systems, thin-client systems usually don’t have complex requirements and have fixed applications. So, it is practical to use the multiple-interface and monolithic-kernel operating systems.

5 Conclusion and future works

In this paper, we describe the multiple-interface and monolithic-kernel operating system. It strikes a proper balance between performance and modularity to offer high adaptability, efficiency and security.

A prototype of multiple-interface and monolithic-kernel operating system has been implemented based on Linux 2.4.17 on the network computer in our lab, with one general interface for common applications, and two specific interfaces respectively for a browser and multimedia. The multiple-interface mechanism can be achieved with few modifications to existing operating system, including interface management facilities and access control mechanisms. Furthermore, unfledged test shows that this operating system introduces little of overhead. More works need to be done, for example, to analyse the complexity and stability of the kernel and the design principle of each interface, to measure and minimize the effects on common applications, to optimize the implementation of the specific interface, and so on.

We believe that the multiple-interface and monolithic-kernel operating system is a better solution to the thin-client platforms than traditional unitary-interface operating systems, enabling them provide efficient and secure services to diverse applications.

6 References

Guan Xuetao: was born in 1974. He received B.S. degree in computer science from Shandong University in 1997. Currently he is a doctor candidate in Micro processor Research and Development Center of Peking University. He has taken part in several projects related to microprocessor and embedded system software. His research interests include operating system, embedded system, system software and system chips.

Liu Shu: was born in 1982. She received B.S. in computer science from Beijing Institute of Technology in 2004. Now she is working toward M.S. degree in Micro Processor Research and Development Center of Peking University. Her research interests include operating system, embedded system and system software.

Cheng Xu: was born in 1964. He is a professor and doctoral supervisor of Perking University. He is also the member of VLSI Design Expert Group of National 863 Program. He received PH.D degree from Harbin Institute of Technology in 1994 and Post Doctor from Computer Science Department of Peking University in 1996. His main research interests are high performance microprocessor Design, SoC design, embedded system, instruction level parallelism, compiler optimization and hardware software codesign.